Prerequisite Concepts for Analysis of Algorithms

Basic Data Structures (Version 7)

Concept: *mathematics notation*

1. $\log_2 n$ is:

   (A) $\Theta(\log_{10} n)$
   (B) $\omega(\log_{10} n)$
   (C) $o(\log_{10} n)$

2. $\log_2 n$ is equal to:

   (A) $\frac{\log_2 n}{\log_2 10}$
   (B) $\frac{\log_2 n}{\log_{10} 2}$
   (C) $\frac{\log_{10} n}{\log_2 10}$
   (D) $\frac{\log_{10} n}{\log_{10} 2}$

3. $\log(nm)$ is equal to:

   (A) $m \log n$
   (B) $\log n + \log m$
   (C) $(\log n)^m$
   (D) $n \log m$

4. $\log(n^m)$ is equal to:

   (A) $n \log m$
   (B) $\log n + \log m$
   (C) $m \log n$
   (D) $(\log n)^m$

5. $\log_2 2$ can be simplified to:

   (A) 1
   (B) $\log_2 2$ cannot be simplified any further
   (C) 2
   (D) 4

6. $2^{\log_2 n}$ is equal to:

   (A) $\log_2 n$
   (B) n
   (C) 2^n
   (D) n^2

7. n^2 is $o(n^3)$. Therefore, $\log n^2$ is $?(\log n^3)$. Choose the tightest bound.

   (A) theta
   (B) big omega
   (C) big omicron
   (D) little omega
   (E) little omicron

8. $\log n^n$ is $\Theta(?)$.

   (A) $\log\ n$
   (B) $n\ \log\ n$
   (C) $\log\ n^{\log n}$
   (D) n

9. $\log 2^n$ is $\Theta(?)$.

   (A) 2^n
   (B) $\log n$
   (C) $n \log n$
   (D) n

10. The number of permutations of a list of n items is:

    (A) $n!$
    (B) $n \log n$
    (C) $\log n$
    (D) n
    (E) 2^n

Concept: relative growth rates

11. Which of the following has the correct order in terms of growth rate?

    (A) $1 < \sqrt{n} < \log n < n < n \log n < n^2 < n^3 < 2^n < n! < n^n$

    (B) $1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < 2^n < n! < n^n$

    (C) $1 < \sqrt{n} < \log n < n < n \log n < n^2 < n^3 < n! < 2^n < n^n$

    (D) $1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < 2^n < n^n < n!$

    (E) $1 < \sqrt{n} < \log n < n < n \log n < n^2 < n^3 < n! < 2^n < n^n$

12. What is the correct ordering of growth rates for the following functions:

    - $f(n) = n^{0.9} \log n$
    - $g(n) = 1.1^n$
    - $h(n) = 9.9n$

    (A) $f < g < h$

    (B) $g < f < h$

    (C) $f < h < g$

    (D) $h < g < f$

    (E) $g < h < f$

    (F) $h < f < g$

13. What is the correct ordering of growth rates for the following functions:

    - $f(n) = n(\log n)^2$
    - $g(n) = n \log 2^n$
    - $h(n) = n \log(\log n)$

    (A) $h > f > g$

    (B) $g > f > h$

    (C) $h > g > f$

    (D) $f > h > g$

    (E) $f > g > h$

    (F) $g > h > f$

Concept: *order notation*

14. What does big Omicron roughly mean?

    (A) worse than or the same as

    (B) better than or the same as

    (C) better than

    (D) the same as

    (E) worse than

15. What does ω roughly mean?

    (A) worse than or the same as

    (B) the same as

    (C) worse than

    (D) better than or the same as

    (E) better than

16. What does θ roughly mean?

    (A) better than or the same as

    (B) worse than

    (C) the same as

    (D) better than

    (E) worse than or the same as

17. **T** or **F**: All algorithms are $\omega(1)$.

18. **T** or **F**: All algorithms are $\theta(1)$.

19. **T** or **F**: All algorithms are $\Omega(1)$.

20. **T** or **F**: There exist algorithms that are $\omega(1)$.

21. **T** or **F**: There exist algorithms that are $O(1)$.

22. **T** or **F**: All algorithms are $O(n^n)$.

23. Consider sorting 1,000,000 numbers with mergesort. What is the time complexity of this operation? [THINK!]

    (A) constant, because n is fixed
    (B) n^2, because mergesort takes quadratic time
    (C) $n \log n$, because mergesort takes $n \log n$ time

Concept: *comparing algorithms using order notation*

Consider the worst case behavior and sufficiently large input size unless otherwise indicated. The phrase *the same time as* means *equal within a constant factor (or lower order term)* unless otherwise indicated. The phrase *by a stopwatch* means the actual amount of time needed for the algorithm to run to completion, as measured by a stopwatch.

24. **T** or **F**: If $f = \Omega(g)$, then algorithm f runs slower than g.

25. **T** or **F**: If $f = \Omega(g)$, then algorithm f runs slower than g, in all cases.

26. **T** or **F**: If $f = \Omega(g)$, then algorithm f runs slower than g, regardless of input size.

27. **T** or **F**: If $f = o(g)$, then algorithm f always runs faster than g.

28. **T** or **F**: If $f = o(g)$, then algorithm f always runs faster than g, in all cases.

29. **T** or **F**: If $f = o(g)$, then algorithm f always runs faster than g, regardless of input size.

30. **T** or **F**: If $f = \Theta(g)$, then algorithm f runs in time equal to g.

31. **T** or **F**: If $f = \Theta(g)$, then algorithm f runs in time equal to g, in all cases.

32. **T** or **F**: If $f = \Theta(g)$, then algorithm f runs in time equal to g, regardless of input size.

33. **T** or **F**: If $f = o(g)$, then algorithm f runs faster than g, regardless of input size.

34. **T** or **F**: If $f = \Omega(g)$, then algorithm f runs faster than or time equal to g, in all cases.

35. **T** or **F**: If $f = O(g)$, then there exists a problem size above which f is always runs faster than or the same time as g, even in the best of cases.

36. **T** or **F**: If $f = O(g)$, then, in the worst case, f always runs faster than or time equal to g.

37. **T** or **F**: If $f = o(g)$, then, in the worst case, there exists a problem size above which f always runs faster than g.

38. **T** or **F**: If $f = \Omega(g)$, then, in the worst case, there exists a problem size above which f always runs slower than g.

39. **T** or **F**: If $f = o(g)$, then, in the worst case, there exists a problem size above which f always runs faster than or equal to g.

40. **T** or **F**: If $f = o(g)$, then there exists a problem size above which f is always runs slower than g, even in the best of cases.

41. **T** or **F**: If $f = \Omega(g)$, then f and g can be the same algorithm.

42. **T** or **F**: If $f = o(g)$, then f and g can be the same algorithm.

43. **T** or **F**: Suppose algorithm $f = \theta(g)$. f and g can be the same algorithm.

44. **T** or **F**: If $f = \Omega(g)$ and $f = O(g)$, then $f = \Theta(g)$.

45. **T** or **F**: Suppose algorithm $f = \theta(g)$. Algorithm f is never slower than g, by a stopwatch.

Concept: *analyzing code*

In the pseudocode, the lower limit of a `for` loop is inclusive, while the upper limit is exclusive. The step, if not specified, is one.

46. What is the time complexity of this function? Assume the initial value of i is zero.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i+1,n);
        }
    }
```

(A) $\theta(n^2)$
(B) $\theta(\log n)$
(C) $\theta(\sqrt{n})$
(D) $\theta(1)$
(E) $\theta(\log n)$
(F) $\theta(n)$

47. What is the time complexity of this function? Assume the initial value of i is one.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i*3,n);
        }
    }
```

(A) $\theta(n^2)$
(B) $\theta(n \log n)$
(C) $\theta(\log n)$
(D) $\theta((\log n)^3)$
(E) $\theta(n)$
(F) $\theta(n\sqrt{n})$

48. What is the time complexity of this function? Assume the initial value of i is one.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i*sqrt(n),n);
        println(i);
        }
    }
```

(A) $\theta(n^2)$
(B) $\theta(1)$
(C) $\theta(\sqrt{n})$
(D) $\theta(\log n)$
(E) $\theta(n)$
(F) $\theta(n\sqrt{n})$

49. What is the time complexity of this function? Assume the initial value of i and j is zero.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < n)
            f(i,j+1,n);
        else
            f(i+1,i+1,n);
        }
    println(i,j);
    }
```

(A) $\theta(n)$
(B) $\theta(n^2)$
(C) $\theta(\log^2 n)$
(D) $\theta(n \log n)$
(E) $\theta(1)$
(F) $\theta(\sqrt{n})$

50. What is the time complexity of this function? Assume the initial value of i and j is zero.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < i)
            f(i,j+1,n);
        else
            f(i+1,0,i+1);
        }
    println(i,j);
    }
```

(A) $\theta(n \log n)$

(B) $\theta(1)$

(C) $\theta(n^2)$

(D) $\theta(n)$

(E) $\theta(\log^2 n)$

(F) $\theta(\sqrt{n})$

51. What is the time complexity of this function? Assume the initial value of i is one and j is zero.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < n)
            f(i,j+1,n);
        else
            f(i*2,0,n);
        }
    println(i,j);
    }
```

(A) $\theta(n^2)$

(B) $\theta(n\sqrt{n})$

(C) $\theta(1)$

(D) $\theta(\log^2 n)$

(E) $\theta(n)$

(F) $\theta(n \log n)$

52. What is the time complexity of this function? Assume the initial value of i is one and j is zero.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < n)
            f(i,j+1,n);
        else
            f(i*2,i*2,n);
        }
    println(i,j);
    }
```

(A) $\theta(n \log n)$

(B) $\theta(1)$

(C) $\theta(\log^2 n)$

(D) $\theta(n\sqrt{n})$

(E) $\theta(n)$

(F) $\theta(n^2)$

53. What is the time complexity of this function? Assume the initial value of i is zero and j is one.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < n)
            f(i,j*2,n);
        else
            f(i+1,1,n);
        }
    println(i,j);
    }
```

(A) $\theta(n\sqrt{n})$

(B) $\theta(1)$

(C) $\theta(n)$

(D) $\theta(n^2)$

(E) $\theta(n \log n)$

(F) $\theta(\log^2 n)$

54. What is the time complexity of this function? Assume positive, integral input and integer division.

```
function f(n)
    {
    if (x > 0)
        {
        f(n/2);
        for (var i from 0 until n)
            println(n);
        }
    }
```

(A) $\theta(n)$

(B) $\theta(\log n)$

(C) $\theta(n^2)$

(D) $\theta(n\sqrt{n})$

(E) $\theta(n \log n)$

(F) $\theta(1)$

55. What is the time complexity of this code fragment?

```
for (i from 0 until n by 1)
    for (j from 0 until i by 1)
        println(i,j);
```

(A) $\theta(n \log n)$

(B) $\theta(\log^2 n)$

(C) $\theta(n)$

(D) $\theta(n^2)$

(E) $\theta(1)$

(F) $\theta(n\sqrt{n})$

56. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    for (j from 0 until n by 1)
        println(i,j);
    i = i * 2;
    }
```

(A) $\theta(1)$

(B) $\theta(n \log n)$

(C) $\theta(n\sqrt{n})$

(D) $\theta(n^2)$

(E) $\theta(n)$

(F) $\theta(\log^2 n)$

57. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    for (j from 0 until i by 1)
        println(i,j);
    i = i * 2;
    }
```

(A) $\theta(\log^2 n)$
(B) $\theta(n)$
(C) $\theta(1)$
(D) $\theta(n\sqrt{n})$
(E) $\theta(n^2)$
(F) $\theta(n \log n)$

58. What is the time complexity of this code fragment?

```
for (i from 0 until n)
    {
    j = 1;
    while (j < n)
        {
        println(i,j);
        j = j * 2;
        }
    }
```

(A) $\theta(n^2)$
(B) $\theta(n\sqrt{n})$
(C) $\theta(1)$
(D) $\theta(n \log n)$
(E) $\theta(\log^2 n)$
(F) $\theta(n)$

59. What is the time complexity of this code fragment?

```
for (i from 0 until n by 1)
    println(i);
```

(A) $\theta(n\sqrt{n})$
(B) $\theta(n)$
(C) $\theta(n \log n)$
(D) $\theta(\log^2 n)$
(E) $\theta(n^{\sqrt{n}})$
(F) $\theta(n^2)$

60. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * 2;
    }
```

(A) $\theta((\log n)^2)$
(B) $\theta(n\sqrt{n})$
(C) $\theta(n)$
(D) $\theta(\log n)$
(E) $\theta(n^2)$
(F) $\theta(n \log n)$

61. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * sqrt(n);
    }
```

(A) $\theta(n)$
(B) $\theta(n\frac{n}{\sqrt{n}})$
(C) $\theta(1)$
(D) $\theta(n\sqrt{n})$
(E) $\theta(n-\sqrt{n})$
(F) $\theta(\sqrt{n})$

62. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * 2;
    }
for (j from 0 until n by 2)
    println(j);
```

(A) $\theta(n)$
(B) $\theta(1)$
(C) $\theta(n^2)$
(D) $\theta(n\log n)$
(E) $\theta(\log^2 n)$
(F) $\theta(n\sqrt{n})$

63. What is the time complexity of this code fragment?

```
for (i from 0 until n by 1)
    println(i);
for (j from 0 until n by 1)
    println(j);
```

(A) $\theta(n\sqrt{n})$
(B) $\theta(n^2)$
(C) $\theta(n)$
(D) $\theta(\log^2 n)$
(E) $\theta(1)$
(F) $\theta(n\log n)$

64. What is the time complexity of this code fragment?

```
for (i from 0 until n by 2)
    println(i);
j = 1;
while (j < n)
    {
    println(j);
    j = j * 2;
    }
```

(A) $\theta(n^2)$
(B) $\theta(\log^2 n)$
(C) $\theta(1)$
(D) $\theta(n\sqrt{n})$
(E) $\theta(n)$
(F) $\theta(n\log n)$

65. What is the space complexity of this code fragment?

```
for (i from 0 until n by 1)
    println(i);
```

(A) $\theta(n^2)$
(B) $\theta(n\log n)$
(C) $\theta(\log n)$
(D) $\theta(n)$
(E) $\theta(\sqrt{n})$
(F) $\theta(1)$

66. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * sqrt(n);
    }
```

(A) $\theta(1)$
(B) $\theta(n)$
(C) $\theta(n \log n)$
(D) $\theta(n^2)$
(E) $\theta(\log n)$
(F) $\theta(\sqrt{n})$

67. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * 2;
    }
```

(A) $\theta(1)$
(B) $\theta(\log n)$
(C) $\theta(\sqrt{n})$
(D) $\theta(n^2)$
(E) $\theta(n \log n)$
(F) $\theta(n)$

68. What is the space complexity of this code fragment?

```
for (i from 0 until n by 1)
    println(i);
for (j from 0 until n by 1)
    println(j);
```

(A) $\theta(n^2)$
(B) $\theta(n \log n)$
(C) $\theta(n)$
(D) $\theta(\log n)$
(E) $\theta(1)$
(F) $\theta(\sqrt{n})$

69. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * 2;
    }
for (j from 0 until n by 2)
    println(j);
```

(A) $\theta(n \log n)$
(B) $\theta(n)$
(C) $\theta(\log n)$
(D) $\theta(1)$
(E) $\theta(\sqrt{n})$
(F) $\theta(n^2)$

70. What is the space complexity of this code fragment?

```
for (i from 0 until n by 2)
    println(i);
j = 1;
while (j < n)
    {
    println(j);
    j = j * 2;
    }
```

(A) $\theta(\log n)$
(B) $\theta(n)$
(C) $\theta(n \log n)$
(D) $\theta(n^2)$
(E) $\theta(1)$
(F) $\theta(\sqrt{n})$

71. What is the space complexity of this code fragment?

```
for (i from 0 until n by 1)
    for (j from 0 until i by 1)
        println(i,j);
```

(A) $\theta(n \log n)$
(B) $\theta(\sqrt{n})$
(C) $\theta(1)$
(D) $\theta(\log n)$
(E) $\theta(n^2)$
(F) $\theta(n)$

72. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    for (j from 0 until i by 1)
        println(i,j);
    i = i * 2;
    }
```

(A) $\theta(n \log n)$
(B) $\theta(n)$
(C) $\theta(n^2)$
(D) $\theta(\sqrt{n})$
(E) $\theta(1)$
(F) $\theta(\log n)$

73. What is the space complexity of this code fragment?

```
for (i from 0 until n)
    {
    j = 1;
    while (j < n)
        {
        println(i,j);
        j = j * 2;
        }
    }
```

(A) $\theta(n)$
(B) $\theta(n^2)$
(C) $\theta(n \log n)$
(D) $\theta(\sqrt{n})$
(E) $\theta(\log n)$
(F) $\theta(1)$

74. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    for (j from 0 until n by 1)
        println(i,j);
    i = i * 2;
    }
```

(A) $\theta(n)$
(B) $\theta(n^2)$
(C) $\theta(n \log n)$
(D) $\theta(\sqrt{n})$
(E) $\theta(1)$
(F) $\theta(\log n)$

75. What is the space complexity of this function? Assume the initial value of i and j is zero.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < n)
            f(i,j+1,n);
        else
            f(i+1,i+1,n);
        }
    println(i,j);
    }
```

(A) $\theta(n^2)$
(B) $\theta(\sqrt{n})$
(C) $\theta(1)$
(D) $\theta(\log^2 n)$
(E) $\theta(n)$
(F) $\theta(n \log n)$

76. What is the space complexity of this function? Assume the initial value of i and j is zero.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < i)
            f(i,j+1,n);
        else
            f(i+1,0,i+1);
        }
    println(i,j);
    }
```

(A) $\theta(n^2)$
(B) $\theta(\sqrt{n})$
(C) $\theta(n \log n)$
(D) $\theta(n)$
(E) $\theta(1)$
(F) $\theta(\log^2 n)$

77. What is the space complexity of this function? Assume the initial value of i is one and j is zero.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < n)
            f(i,j+1,n);
        else
            f(i*2,0,n);
        }
    println(i,j);
    }
```

(A) $\theta(n)$
(B) $\theta(n \log n)$
(C) $\theta(1)$
(D) $\theta(n^2)$
(E) $\theta(\log^2 n)$
(F) $\theta(n\sqrt{n})$

78. What is the space complexity of this function? Assume the initial value of i is one and j is zero.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < n)
            f(i,j+1,n);
        else
            f(i*2,i*2,n);
        }
    println(i,j);
    }
```

(A) $\theta(1)$
(B) $\theta(n\sqrt{n})$
(C) $\theta(n^2)$
(D) $\theta(\log^2 n)$
(E) $\theta(n \log n)$
(F) $\theta(n)$

79. What is the space complexity of this function? Assume the initial value of i is zero and j is one.

```
function f(i,j,n)
    {
    if (i < n)
        {
        if (j < n)
            f(i,j*2,n);
        else
            f(i+1,1,n);
        }
    println(i,j);
    }
```

(A) $\theta(1)$
(B) $\theta(\log^2 n)$
(C) $\theta(n \log n)$
(D) $\theta(n^2)$
(E) $\theta(n\sqrt{n})$
(F) $\theta(n)$

80. What is the space complexity of this function? Assume positive, integral input and integer division.

```
function f(x,n)
    {
    if (x > 0)
        {
        f(x/2,n);
        for (var i from 0 until n)
            println(n);
        }
    }
```

(A) $\theta(1)$
(B) $\theta(\log n)$
(C) $\theta(n\sqrt{n})$
(D) $\theta(n^2)$
(E) $\theta(n)$
(F) $\theta(n \log n)$

81. What is the space complexity of this function? Assume the initial value of i is zero.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i+1,n);
        println(i);
        }
    }
```

(A) $\theta(1)$
(B) $\theta(n \log n)$
(C) $\theta(n^2)$
(D) $\theta(n)$
(E) $\theta(\log n)$
(F) $\theta(\sqrt{n})$

82. What is the space complexity of this function? Assume the initial value of i is one.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i*2,n);
        println(i);
        }
    }
```

(A) $\theta(\log n)$
(B) $\theta(n \log n)$
(C) $\theta(\sqrt{n})$
(D) $\theta(1)$
(E) $\theta(n^2)$
(F) $\theta(n)$

83. What is the space complexity of this function? Assume the initial value of i is one.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i*sqrt(n),n);
        println(i);
        }
    }
```

(A) $\theta(\sqrt{n})$
(B) $\theta(n\sqrt{n})$
(C) $\theta(n\frac{n}{\sqrt{n}})$
(D) $\theta(1)$
(E) $\theta(n)$
(F) $\theta(n - \sqrt{n})$

84. What is the space complexity of this function? Assume the initial value of i is one.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i+sqrt(n),n);
        println(i);
        }
    }
```

(A) $\theta(n)$
(B) $\theta(1)$
(C) $\theta(\sqrt{n})$
(D) $\theta(n-\sqrt{n})$
(E) $\theta(n\frac{n}{\sqrt{n}})$
(F) $\theta(n\sqrt{n})$

Concept: *analysis of classic, simple algorithms*

85. Which of the following describes the classic recursive fibonacci's time complexity?

(A) $\theta(\Phi)$
(B) $\theta(\sqrt{n})$
(C) $\theta(\frac{n}{\sqrt{n}})$
(D) $\theta(\Phi^n)$
(E) $\theta(n-\sqrt{n})$
(F) $\theta(1)$
(G) $\theta(\frac{\Phi}{n})$

86. Which of the following describes the classic recursive fibonacci's space complexity?

(A) $\theta(\sqrt{n})$
(B) $\theta(1)$
(C) $\theta(n)$
(D) $\theta(\frac{\Phi}{n})$
(E) $\theta(\frac{n}{\sqrt{n}})$
(F) $\theta(n-\sqrt{n})$

87. Which of the following describes iterative actorial's. time complexity?

(A) $\theta(n-\sqrt{n})$
(B) $\theta(1)$
(C) $\theta(\frac{n}{\sqrt{n}})$
(D) $\theta(\frac{\Phi}{n})$
(E) $\theta(\sqrt{n})$
(F) $\theta(n)$

88. Which of the following describes iterative fibonacci's space complexity?

(A) $\theta(n-\sqrt{n})$
(B) $\theta(n)$
(C) $\theta(\frac{\Phi}{n})$
(D) $\theta(\sqrt{n})$
(E) $\theta(\frac{n}{\sqrt{n}})$
(F) $\theta(1)$

Concept: searching

89. **T** or **F**: The following code reliably sets the variable *min* to the minimum value of an unsorted, non-empty array.

```
min = 0;
for (i from 0 until array.length)
    if (array[i] < min)
        min = array[i];
```

90. **T** or **F**: The following code reliably sets the variable *max* to the maximum value in an unsorted, non-empty array.

```
max = array[0]
for (i from 0 to array.length)
    if (array[i] > max)
        max = array[i]
```

91. **T** or **F**: The following function reliably returns `True` if the value of item is *present* in the unsorted, non-empty array.

```
function find(array,item)
    {
    found = False;
    for (i from 0 until array.length)
        if (array[i] == item)
            found = True;
    return found;
    }
```

92. **T** or **F**: The following function reliably returns `False` if the value of item is *missing* in the unsorted, non-empty array.

```
function find(array,item)
    {
    found = True;
    for (i from 0 unitl array.length)
        if (array[i] != item)
            found = False;
    return found;
    }
```

93. What is the average and worst case time complexity, respectively, for searching an unordered list?

(A) linear, log,
(B) linear, linear
(C) log, linear
(D) log, log

94. What is the average and worst case time complexity, respectively, for searching an ordered list?

(A) linear, linear
(B) log, log
(C) log, linear
(D) linear, log

Concept: sorting

95. The following strategy is employed by which sort: *find the most extreme value in the unsorted portion and place it at the boundary of the sorted and unsorted portions*?

(A) selection sort
(B) mergesort
(C) heapsort
(D) bubble sort
(E) insertion sort
(F) quicksort

96. The following strategy is employed by which sort: *sort the lower half of the items to be sorted, then sort the upper half, then arrange things such that the largest item in the lower half is less than or equal to the smallest item in the upper half* ?

(A) mergesort
(B) insertion sort
(C) selection sort
(D) bubble sort
(E) heapsort
(F) quicksort

97. The following strategy is employed by which sort: *take the first value in the unsorted portion and place it where it belongs in the sorted portion*?

(A) insertion sort
(B) heapsort
(C) selection sort
(D) mergesort
(E) bubble sort
(F) quicksort

98. The following strategy is employed by which sort: *pick a value and arrange things such that the largest item in the lower portion is less than or equal to the value and that the smallest item in the upper portion is greater than or equal to the value, then sort the lower portion, then sort the upper* ?

(A) selection sort
(B) bubble sort
(C) heapsort
(D) quicksort
(E) mergesort
(F) insertion sort

99. Which sort optimizes the worst case behavior of bubble sort?

(A) insertion sort
(B) heapsort
(C) quicksort
(D) stooge sort
(E) mergesort
(F) selection sort

Concept: *space and time complexity*

100. What is the best time case complexity for mergesort?

(A) $n \log n$
(B) cubic
(C) linear
(D) quadratic
(E) $\log n$

101. What is the worst case complexity for classical mergesort?

(A) $\log n$
(B) cubic
(C) $n \log n$
(D) linear
(E) quadratic

102. If quicksort is implemented such that the pivot is chosen to be the first element in the array, the worst case behavior of the sort is:

(A) log linear
(B) quadratic
(C) linear
(D) exponential

103. If quicksort is implemented such that the a random element is chosen to be the pivot, the worst case behavior of the sort is:

(A) exponential
(B) linear
(C) log linear
(D) quadratic

104. What is the best case complexity for quicksort?

(A) quadratic
(B) cubic
(C) $\log n$
(D) linear
(E) $n \log n$

105. What is the best case complexity for selection sort?

(A) linear
(B) cubic
(C) $n \log n$
(D) $\log n$
(E) quadratic

106. What is the worst case complexity for classical selection sort?

(A) quadratic
(B) cubic
(C) $n \log n$
(D) $\log n$
(E) linear

107. What is the best case complexity for insertion sort?

(A) linear
(B) quadratic
(C) cubic
(D) $n \log n$
(E) $\log n$

108. What is the worst case complexity for classical insertion sort?

(A) $n \log n$
(B) quadratic
(C) linear
(D) cubic
(E) $\log n$

Concept: *simple arrays*

Assume zero-based indexing for all arrays.

In the pseudocode, the lower limit of a `for` loop is inclusive, while the upper limit is exclusive. The step, if not specified, is one.

For all types of fillable arrays, the size is the number of elements added to the array; the capacity is the maximum number of elements that can be added to the array.

109. Consider a small array a and large array b. Accessing the element in the first slot of a takes more/less/the same amount of time as accessing the element in the first slot of b.

(A) it depends on how the arrays were allocated
(B) more time
(C) the same amount of time
(D) less time

110. Consider a small array a and large array b. Accessing the element in the last slot of a b takes more than/less than/the same amount of time as accessing an element in the middle slot of a. Both indices are supplied.

(A) the same amount of time
(B) more time
(C) less time
(D) it depends on how the arrays were allocated

111. Accessing the middle element of an array takes more/less/the same amount of time than accessing the last element.

(A) the same amount of time
(B) more time
(C) less time
(D) it depends on how the array were allocated

112. What is a major characteristic of a simple array?

(A) finding an element can be done in constant time
(B) inserting an element between indices i and $i+1$can be done in constant time
(C) getting the value at an index can be done in constant time

113. What is a *not* a major characteristic of a simple array?

(A) getting the value at an index can be done in constant time
(B) finding an element can be done in constant time
(C) setting the value at an index can be done in constant time
(D) swapping two elements can be done in constant time

114. Does the following code set the variable v to the minimum value in an unsorted array with at least two elements?

```
v = 0;
for (i from 0 until array.length)
    if (array[i] < v)
        v = array[i];
```

(A) only if all elements have the same value
(B) never
(C) yes, if all the elements are positive
(D) yes, if all the elements are negative
(E) only if the true minimum value is zero
(F) always

115. Does the following code set the variable v to the minimum value in an unsorted array with at least two elements?

```
v = array[0];
for (i from 0 until array.length)
    if (array[i] < v)
        v = array[i];
```

(A) yes, if all the elements are negative
(B) only if the true minimum value is at index 0
(C) always
(D) never
(E) only if all elements have the same value
(F) yes, if all the elements are positive

116. Does the following code set the variable v to the minimum value in an unsorted array with at least two elements?

```
v = array[0];
for (i from 0 until array.length)
    if (array[i] > v)
        v = array[i];
```

(A) yes, if all the elements are positive
(B) only if all elements have the same value
(C) only if the true minimum value is at index 0
(D) yes, if all the elements are negative
(E) never
(F) always

117. Does the following code set the variable v to the minimum value in an unsorted, non-empty array?

```
v = array[0];
for (i from 0 until array.length)
    if (array[i] > v)
        v = array[i];
```

(A) always
(B) never
(C) only if the true minimum value is at index 0
(D) yes, if all the elements are negative
(E) yes, if all the elements are positive
(F) only if all elements have the same value

118. Does this *find* function return the expected result? Assume the array has at least two elements.

```
function find(array,item)
    {
    var i; var found = False;
    for (i from 0 until array.length)
        if (array[i] == item)
            found = True;
    return found;
    }
```

(A) only if the item is not in the array
(B) always
(C) never
(D) only if the item is in the array

119. Does this *find* function return the expected result? Assume the array has at least two elements.

```
function find(array,item)
    {
    var i;
    for (i from 0 until array.length)
        if (array[i] == item)
            return False;
    return True;
    }
```

(A) only if the item is in the array
(B) only if the item is not in the array
(C) always
(D) never

120. Is this *find* function correct? Assume the array has at least two elements.

```
function find(array,item)
    {
    var i; var found = True;
    for (i from 0 until array.length)
        if (array[i] != item)
            found = False;
    return found;
    }
```

(A) always

(B) never

(C) only if the item is not in the array

(D) only if the item is in the array

121. Does this *find* function return the expected result? Assume the array has at least two elements.

```
function find(array,item)
    {
    var i;
    for (i from 0 until array.length)
        if (array[i] == item)
            return True;
    return False;
    }
```

(A) only if the item is not in the array

(B) always

(C) only if the item is in the array

(D) never

Concept: *simple fillable arrays*

Assume the back index in a simple fillable array points to the first available slot.

122. What is *not* a property of a simple fillable array?

(A) elements are presumed to be contiguous

(B) the underlying simple array can increase in size

(C) there exists an element that can be removed in constant time

(D) elements can be added in constant time

123. What is a property of a simple fillable array?

(A) any element can be removed in constant time

(B) elements are presumed to be contiguous

(C) more that one element can be next to an empty slot

(D) an element can be added anywhere in constant time

124. Adding an element at back of a simple fillable array can be done in:

(A) quadratic time

(B) constant time

(C) logarithmic time

(D) linear time

125. Removing an element at front of a simple fillable array can be done in:

(A) quadratic time

(B) linear time

(C) logarithmic time

(D) constant time

126. Suppose a simple fillable array has size s and capacity c. The next value to be added to the array will be placed at index:

(A) c - 1

(B) c

(C) $c + 1$

(D) s

(E) $s + 1$

(F) s - 1

127. Suppose for a simple fillable array, the size is one less than the capacity. How many values can still be added?

(A) one
(B) this situation cannot exist
(C) zero, the array is full
(D) two

128. Suppose for a simple fillable array, the capacity is one less than the size. How many values can still be added?

(A) one
(B) this situation cannot exist
(C) two
(D) zero, the array is full

129. Suppose a simple fillable array is empty. The size of the array is:

(A) the length of the underlying simple array
(B) the capacity of the array
(C) one
(D) zero

130. Suppose a simple fillable array is full. The capacity of the array is:

(A) zero
(B) one
(C) the length of the underlying simple array
(D) its size minus one

131. Which code fragment correctly inserts a new element into index j of a simple fillable array with size s? Assume there is room for the new element.

```
for (i from j until s-2)
    array[i] = array[i+1];
array[i] = newElement;
---
for (i from s-2 until j)
    array[i+1] = array[i];
array[i] = newElement;
```

(A) the first fragment
(B) neither are correct
(C) the second fragment
(D) both are correct

132. Which code fragment correctly inserts a new element into index j of an array with size s?

```
for (i from j until s-2)
    array[i+1] = array[i];
array[i] = newElement;
---
for (i from s-2 until j)
    array[i] = array[i+1];
array[i] = newElement;
```

(A) the first fragment
(B) the second fragment
(C) both are correct
(D) neither are correct

Concept: *circular arrays*

For circular arrays, assume f is the start index, e is the end index, s is the size, and c is the capacity of the array. Both f and e point to the first available slots.

133. What is a property of a theoretical (not practical) circular array?

(A) an element can be added anywhere in constant time
(B) elements do not have to be contiguous
(C) there are two places an element can be added
(D) any element can be removed in constant time

134. What is *not* a property of a theoretical (not practical) circular array?

(A) appending an element can be done in constant time
(B) prepending an element can be done in constant time
(C) inserting an element in the middle can be done in constant time
(D) elements are presumed to be contiguous

135. The next value to be added to the front of a circular array will be placed at index:

(A) c - 1
(B) f
(C) s - f
(D) $s + f$
(E) f - 1
(F) c - f

136. Suppose for a circular array, the size is equal to the capacity. Can a value be added?

(A) No, the array is completely full
(B) Yes, there is room for one more value

137. Suppose a circular array is empty. The size of the array is:

(A) one
(B) the length of the array
(C) the capacity of the array
(D) zero

138. In a circular array, which is *not* a proper way to correct the start index f after an element is added to the front of the array?

(A) `f = f == 0? c - 1 : f - 1;`
(B) `if (f == 0) f = c - 1; else f = f - 1;`
(C) `f -= 1; f = f < 0? c - 1 : f;`
(D) `f -= 1; f == 0? c - 1 : f;`
(E) `f = (f - 1 + c) % c;`

139. **T** or **F**: In a circular array, the start index (after correction) can never equal the size of the array.

140. **T** or **F**: In a circular array, the start index (after correction) can never equal the capacity of the array.

141. Is a separate end index e needed in a circular array?

(A) no, it can be computed from c and f.
(B) no, it can be computed from s and f.
(C) no, it can be computed from s, c, and f.
(D) no, it can be computed from s and c.
(E) yes

Concept: *dynamic arrays*

142. What is *not* a major characteristic of a dynamic array?

(A) elements are presumed to be contiguous
(B) the array can grow to accommodate more elements
(C) inserting an element in the middle takes linear time
(D) finding an element takes at most linear time
(E) the only allowed way to grow is doubling the size

143. Suppose a dynamic array has size s and capacity c, with s equal to c. Is the array required to grow on the next addition?

(A) yes, but only if the dynamic array is not circular
(B) no
(C) yes

144. Suppose array capacity grows by 50<percent/> every time a dynamic array fills. If the only events are insertions, the growing events:

(A) occur less and less frequently
(B) cannot be characterized in terms of frequency
(C) occur more and more frequently
(D) occur periodically

145. Suppose array capacity doubles every time a dynamic array fills, If the only events are insertions, the average cost of an insertion, in the limit, is:

(A) the log of the capacity
(B) constant
(C) linear
(D) the log of the size

146. Suppose array capacity grows by 10 every time a dynamic array fills, If the only events are insertions, the average cost of an insertion, in the limit, is:

(A) quadratic
(B) constant
(C) the log of the size
(D) linear
(E) the log of the capacity

147. Suppose array capacity grows by 10 every time a dynamic array fills, If the only events are insertions, the growing events:

(A) occur less and less frequently
(B) occur periodically
(C) cannot be characterized in terms of frequency
(D) occur more and more frequently

148. If array capacity grows by 10 every time a dynamic array fills, the average cost of an insertion in the limit is:

(A) linear
(B) the log of the capacity
(C) the log of the size
(D) constant

Concept: *singly-linked lists (insertions)*

149. Appending to a singly-linked list without a tail pointer takes:

(A) $n \log n$ time
(B) constant time
(C) linear time
(D) log time

150. Appending to a singly-linked list with a tail pointer takes:

(A) constant time
(B) $n \log n$ time
(C) log time
(D) linear time

151. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just prior in:

(A) log time
(B) $n \log n$ time
(C) linear time
(D) constant time

152. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just after in:

(A) $n \log n$ time
(B) log time
(C) constant time
(D) linear time

153. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just after with as few pointer assignments as:

(A) 5
(B) 3
(C) 2
(D) 4
(E) 1

Concept: *singly-linked lists (deletions)*

154. Removing the first item from a singly-linked list without a tail pointer takes:

(A) log time
(B) constant time
(C) $n \log n$ time
(D) linear time

155. Removing the last item from a singly-linked list with a tail pointer takes:

(A) log time
(B) $n \log n$ time
(C) linear time
(D) constant time

156. Removing the last item from a singly-linked list without a tail pointer takes:

(A) log time
(B) linear time
(C) $n \log n$ time
(D) constant time

157. Removing the first item from a singly-linked list with a tail pointer takes:

(A) log time
(B) $n \log n$ time
(C) linear time
(D) constant time

158. In a singly-linked list, you can move the tail pointer back one node in:

(A) constant time
(B) $n \log n$ time
(C) linear time
(D) log time

159. Suppose you have a pointer to a node in the middle of a singly-linked list. You can then delete that node in:

(A) log time
(B) constant time
(C) $n \log n$ time
(D) linear time

Concept: *doubly-linked lists (insertions)*

160. Appending to a non-circular, doubly-linked list without a tail pointer takes:

(A) $n \log n$ time
(B) linear time
(C) constant time
(D) log time

161. Appending to a non-circular, doubly-linked list with a tail pointer takes:

(A) linear time
(B) constant time
(C) log time
(D) $n \log n$ time

162. Removing the first item from a non-circular, doubly-linked list without a tail pointer takes:

(A) constant time
(B) linear time
(C) $n \log n$ time
(D) log time

163. Suppose you have a pointer to a node in the middle of a doubly-linked list. You can then insert a new node just after in:

(A) linear time
(B) constant time
(C) log time
(D) $n \log n$ time

164. Suppose you have a pointer to a node in the middle of a doubly-linked list. You can then insert a new node just prior with as few pointer assignments as:

(A) 3
(B) 5
(C) 1
(D) 4
(E) 2

165. **T** : **F**: Making a doubly-linked list circular removes the need for a separate tail pointer.

Concept: *doubly-linked lists (deletions)*

166. Removing the first item from a doubly-linked list with a tail pointer takes:

(A) $n \log n$ time
(B) linear time
(C) log time
(D) constant time

167. In a doubly-linked list, you can move the tail pointer back one node in:

(A) log time
(B) $n \log n$ time
(C) linear time
(D) constant time

168. In a doubly-linked list, what does a tail-pointer gain you?

(A) the ability to append the list in constant time
(B) the ability to both prepend and remove the first element of list in constant time
(C) the ability to both append and remove the last element of list in constant time
(D) the ability to prepend the list in constant time
(E) the ability to remove the first element of list in constant time
(F) the ability to remove the last element of list in constant time

Concept: *input-output order*

169. These values are pushed onto a stack in the order given: 1 5 9. A *pop* operation would return which value?

(A) 5
(B) 1
(C) 9

170. LIFO ordering is the same as:

(A) LILO
(B) FILO
(C) FIFO

Concept: *time and space complexity*

171. Consider a stack based upon a fillable array with pushes onto the back of the array. What is the time complexity of the worst case behavior for *push* and *pop*, respectively? You may assume there is sufficient space for the *push* operation.

(A) constant and constant
(B) constant and linear
(C) linear and constant
(D) linear and linear

172. Consider a stack based upon a circular array with pushes onto the front of the array. What is the time complexity of the worst case behavior for *push* and *pop*, respectively? You may assume there is sufficient space for the *push* operation.

(A) linear and linear
(B) linear and constant
(C) constant and constant
(D) constant and linear

173. Consider a stack based upon a dynamic array with pushes onto the back of the array. What is the time complexity of the worst case behavior for *push* and *pop*, respectively? You may assume there is sufficient space for the *push* operation and that the array never shrinks.

(A) constant and constant
(B) linear and linear
(C) constant and linear
(D) linear and constant

174. Consider a stack based upon a dynamic array with pushes onto the back of the array. What is the time complexity of the worst case behavior for *push* and *pop*, respectively? You may assume there is sufficient space for the *push* operation and that the array may shrink.

(A) linear and linear
(B) linear and constant
(C) constant and linear
(D) constant and constant

175. Consider a stack based upon a dynamic circular array with pushes onto the front of the array. What is the time complexity of the worst case behavior for *push* and *pop*, respectively? You may assume the array may grow or shrink.

(A) constant and linear
(B) linear and constant
(C) linear and linear
(D) constant and constant

176. Consider a stack based upon a singly-linked list without a tail pointer with pushes onto the front of the list. What is the time complexity of the worst case behavior for *push* and *pop*, respectively?

(A) linear and constant
(B) linear and linear
(C) constant and linear
(D) constant and constant

177. Consider a stack based upon a singly-linked list with a tail pointer with pushes onto the front of the list. What is the time complexity of the worst case behavior for *push* and *pop*, respectively?

(A) linear and constant
(B) linear and linear
(C) constant and linear
(D) constant and constant

178. Consider a stack based upon a non-circular, doubly-linked list without a tail pointer with pushes onto the front of the list. What is the time complexity of the worst case behavior for *push* and *pop*, respectively?

(A) constant and linear
(B) constant and constant
(C) linear and linear
(D) linear and constant

179. Consider a stack based upon a doubly-linked list with a tail pointer with pushes onto the front of the list. What is the time complexity of the worst case behavior for *push* and *pop*, respectively?

(A) constant and linear
(B) linear and linear
(C) linear and constant
(D) constant and constant

180. Suppose a simple fillable array with capacity c is used to implement two stacks, one growing from each end. The stack sizes at any given time are stored in i and j, respectively. If maximum space efficiency is desired, a reliable condition for the stacks being full is:

(A) `i == c/2-1 || j == c/2-1`
(B) `i + j == c-2`
(C) `i == c/2 || j == c/2`
(D) `i + j == c`
(E) `i == c/2-1 && j == c/2-1`
(F) `i == c/2 && j == c/2`

Concept: *stack applications*

For the following questions, assume the tokens in a post-fix equation are processed with the following code, with all functions having their obvious meanings and integer division.

```
s.push(readEquationToken());
s.push(readEquationToken());
while (moreEquationTokens())
    {
    t = readEquationToken();
    if (isNumber(t))
        s.push(t);
    else /* t must be an operator */
        {
        operandB = s.pop();
        operandA = s.pop();
        result = performOperation(t,operandA,operandB);
        s.push(result);
        }
    }
```

181. If the tokens of the postfix equation `8 2 3 ^ / 2 3 * + 5 1 * -` are read in the order given, what are the top two values in s immediately after the result of the first multiplication is pushed?

(A) 1 2
(B) 1 6
(C) 3 3
(D) 5 6

Concept: *input-output order*

182. These values are enqueued onto a queue in the order given: 1 5 9 4. A dequeue operation would return which value?

   (A) 4
   (B) 1
   (C) 5
   (D) 9

183. FIFO ordering is the same as:

   (A) LIFO
   (B) LILO
   (C) FILO

Concept: *complexity*

184. Consider a queue based upon a simple fillable array with enqueues onto the front of the array. What is the time complexity of the worst case behavior for *enqueue* and *dequeue*, respectively? Assume there is room for the operations.

   (A) linear and constant
   (B) constant and linear
   (C) constant and constant
   (D) linear and linear

185. Consider a queue based upon a circular array with enqueues onto the front of the array. What is the time complexity of the worst case behavior for *enqueue* and *dequeue*, respectively? Assume there is room for the operations.

   (A) constant and constant
   (B) constant and linear
   (C) linear and constant
   (D) linear and linear

186. Consider a queue based upon a singly-linked list without a tail pointer with enqueues onto the front of the list. What is the time complexity of the worst case behavior for *enqueue* and *dequeue*, respectively?

   (A) linear and linear
   (B) constant and linear
   (C) constant and constant
   (D) linear and constant

187. Consider a queue based upon a singly-linked list with a tail pointer with enqueues onto the front of the list. What is the time complexity of the worst case behavior for *enqueue* and *dequeue*, respectively?

   (A) linear and linear
   (B) linear and constant
   (C) constant and linear
   (D) constant and constant

188. Consider a queue based upon a doubly-linked list with a tail pointer with enqueues onto the front of the list. What is the time complexity of the worst case behavior for *enqueue* and *dequeue*, respectively?

   (A) constant and linear
   (B) linear and constant
   (C) linear and linear
   (D) constant and constant

189. Consider a queue based upon a non-circular, doubly-linked list without a tail pointer with enqueues onto the front of the list. What is the time complexity of the worst case behavior for *enqueue* and *dequeue*, respectively?

   (A) linear and linear
   (B) constant and constant
   (C) linear and constant
   (D) constant and linear

Concept: *complexity*

190. Consider a worst-case binary search tree with n nodes. What is the average case time complexity for finding a value at a leaf?

   (A) constant
   (B) $n \log n$
   (C) $\sqrt{n}$
   (D) linear
   (E) $\log n$
   (F) quadratic

191. Consider a binary search tree with n nodes. What is the worst case time complexity for finding a value at a leaf?

(A) constant
(B) log n
(C) quadratic
(D) $\sqrt{n}$
(E) n log n
(F) linear

192. Consider a binary search tree with n nodes. What is the minimum and maximum height (using order notation)?

(A) constant and linear
(B) linear and linear
(C) log n and log n
(D) constant and log n
(E) log n and linear

Concept: *balance*

193. Which ordering of input values builds the most unbalanced BST? Assume values are inserted from left to right.

(A) 1 2 3 4 5 7 6
(B) 4 3 1 6 2 8 7
(C) 1 7 2 6 3 5 4

194. Which ordering of input values builds the most balanced BST? Assume values are inserted from left to right.

(A) 4 3 1 6 2 8 7
(B) 1 2 7 6 0 3 8
(C) 1 4 3 2 5 7 6

Concept: *tree shapes*

195. What is the best definition of a perfect binary tree?

(A) all leaves have zero children
(B) all nodes have zero or two children
(C) all null children are equidistant from the root
(D) all leaves are equidistant from the root

196. Suppose a binary tree has 10 leaves. How many nodes in the tree must have two children?

(A) no limit
(B) 10
(C) 9
(D) 7
(E) 8

197. Suppose a binary tree has 10 nodes. How many nodes are children of some other node in the tree?

(A) 7
(B) 9
(C) no limit
(D) 10
(E) 8

198. Let P0, P1, and P2 refer to nodes that have zero, one or two children, respectively. Using the generally accepted definition, what is a *full* binary tree?

(A) all interior nodes are P2; all leaves are equidistant from the root
(B) all interior nodes P1, except the root
(C) all leaves are equidistant from the root
(D) all nodes are P0 or P2
(E) all interior nodes are P1
(F) all interior nodes are P2

199. Let P0, P1, and P2 refer to nodes that have zero, one or two children, respectively. Using the generally accepted definition, what is a *perfect* binary tree?

(A) all interior nodes P1, except the root
(B) all interior nodes are P1
(C) all nodes are P0 or P2
(D) all interior nodes are P2; all leaves are equidistant from the root
(E) all interior nodes are P2
(F) all leaves are equidistant from the root

200. Let P0, P1, and P2 refer to nodes that have zero, one or two children, respectively. Using the generally accepted definition, what is a *degenerate* binary tree?

(A) all leaves are equidistant from the root
(B) all interior nodes are P1
(C) all interior nodes are P2; all leaves are equidistant from the root
(D) all interior nodes P1, except the root
(E) all interior nodes are P2
(F) all nodes are P0 or P2

201. Let P0, P1, and P2 refer to nodes that have zero, one or two children, respectively. Using the generally accepted definition of a *complete* binary tree, which of the following actions can be used to make any complete tree? Assume the leftmost and rightmost sets may be empty.

(A) making the leftmost leaves of a *perfect* tree P2
(B) another name for a *perfect* tree
(C) making the leftmost leaf of a *perfect* tree P1
(D) making the leftmost leaves of a *perfect* tree P1 or P2
(E) removing the rightmost leaves from a *perfect* tree

202. **T** or **F**: All *perfect* trees are *full* trees.

203. **T** or **F**: All *full* trees are *complete* trees.

204. **T** or **F**: All *complete* trees are *perfect* trees.

205. How many distinct binary trees can be formed from two nodes with values 1, 2, or 3 respectively (hint: think about how many permutations of values there are for each tree shape)?

(A) 1
(B) 4
(C) 5
(D) 2
(E) 3

206. How many distinct binary tree shapes can be formed from two nodes?

(A) 4
(B) 3
(C) 2
(D) 1
(E) 5

207. Let k be the the number of steps from the root to a leaf in a perfect tree. What are the number of nodes in the tree?

(A) 2^{k+1}
(B) $2^{k-1}+1$
(C) $2^{k+1}-1$
(D) $2^{k}-1$
(E) $2^{k-1}-1$

208. Let k be the the number of steps from the root to the furthest leaf in a binary tree. What would be the minimum number of nodes in such a tree? Assume k is a power of two.

(A) $2^{k+1}-1$
(B) k
(C) 2^{k+1}
(D) $(\log k)+1$
(E) $k+1$
(F) $\log k$

209. Let k be the the number of steps from the root to the furthest leaf in a binary tree. What would be the maximum number of nodes in such a tree? Assume k is a power of two.

(A) 2^{k+1}
(B) $\log k$
(C) k
(D) $(\log k)+1$
(E) $k+1$
(F) $2^{k+1}-1$

Concept: *ordering in a BST*

210. For all child nodes in a BST, what relationship holds between the value of a left child node and the value of its parent? Assume unique values.

(A) greater than
(B) less than
(C) there is no relationship

211. For all sibling nodes in a BST, what relationship holds between the value of a left child node and the value of its sibling? Assume unique values.

(A) greater than
(B) there is no relationship
(C) less than

212. Which statement is true about the *successor* of a node in a BST, if it exists?

(A) has no left child
(B) it is always a leaf node
(C) it is always an interior node
(D) has no right child
(E) it may be an ancestor

213. Consider a node which holds neither the smallest or the largest value in a BST. Which statement is true about the node which holds the next higher value of a node in a BST, if it exists?

(A) it is always an interior node
(B) has no left child
(C) it is always a leaf node
(D) it may be an ancestor
(E) has no right child

Concept: *traversals*

214. Consider a binary tree with 25 nodes to the left of the root and 38 nodes to the right. How many nodes are processed before the root in a pre-order traversal?

(A) 53
(B) 25
(C) none of the other answers are correct
(D) 54
(E) 38
(F) 0

215. Consider a binary tree with 25 nodes to the left of the root and 38 nodes to the right. How many nodes are processed before the root in an in-order traversal?

(A) none of the other answers are correct
(B) 38
(C) 25
(D) 53
(E) 0
(F) 54

216. Consider a binary tree with 25 nodes to the left of the root and 38 nodes to the right. How many nodes are processed before the root in a post-order traversal?

(A) 63
(B) 0
(C) 25
(D) 38
(E) none of the other answers are correct
(F) 54

217. Consider a perfect BST with even values 0 through 12, to which the value 7 is then added. Which of the following is an in-order traversal of the resulting tree?

(A) 0 4 2 7 8 10 12 6
(B) 0 2 4 6 8 10 12 7
(C) 7 0 2 4 6 8 10 12
(D) 12 10 8 7 6 4 2 0
(E) 6 2 10 0 4 8 12 7
(F) 0 2 4 6 7 8 10 12

218. Consider a perfect BST with even values 0 through 12, to which the value 7 is then added. Which of the following is a level-order traversal of the resulting tree?

(A) 12 10 8 7 6 4 2 0
(B) 0 2 4 6 8 10 12 7
(C) 6 2 10 0 4 8 12 7
(D) 0 2 4 6 7 8 10 12
(E) 7 0 2 4 6 8 10 12
(F) 0 4 2 7 8 10 12 6

219. If a six-node binary tree has a level-order traversal of C B A D F E and an in-order traversal of B C A F D E does it have a unique pre-order traversal and, if so, what is it?

(A) yes, C A D B E F
(B) yes, but the correct answer is not listed
(C) no
(D) yes, C B A D F E
(E) yes, C B A F D E

220. If a six-node binary tree has a in-order traversal of B C A F D E has a pre-order traversal of C B A D F E does it have a unique post-order traversal and, if so, what is it?

(A) yes, B F A E D C
(B) yes, but the correct answer is not listed
(C) yes, F A D B E C
(D) yes, B F E D A C
(E) no

221. If a six-node binary tree has a in-order traversal of B C A F D E has a post-order traversal of C B A D F E does it have a unique level-order traversal and, if so, what is it?

(A) no
(B) yes, but the correct answer is not listed
(C) yes, E F C D B A
(D) yes, E F A D B C
(E) yes, E A C F B D

222. If a six-node binary tree has a level-order traversal of C F D E B A and an pre-order traversal of C F E A D B does it have a unique in-order traversal and, if so, what is it?

(A) yes, F A E C B D
(B) no
(C) yes, but the correct answer is not listed
(D) yes, F E A C D B
(E) yes, F A E C B D

Concept: *insertion and deletion*

223. **T** or **F**: Suppose you are given an in-order traversal of an unbalanced BST. If you were to insert those values into an empty BST in the order given, would the result be a balanced tree?

224. **T** or **F**: Suppose you are given a pre-order traversal of an unbalanced BST. If you were to insert those values into an empty BST in the order given, would the result be a balanced tree?

225. **T** or **F**: Suppose you are given an in-order traversal of a balanced BST. If you were to insert those values into an empty BST in the order given, would the result be a balanced tree?

226. **T** or **F**: Suppose you are given a pre-order traversal of a balanced BST. If you were to insert those values into an empty BST in the order given, would the result be a balanced tree?

227. Suppose 10 values are inserted inserted into an empty BST. What is the minimum and maximum resulting heights of the tree? The height is the number of steps from the root to the furthest leaf.

(A) 4 and 10
(B) 5 and 10
(C) 3 and 10
(D) 5 and 9
(E) 3 and 9
(F) 4 and 9

228. Which, if any, of these deletion strategies for non-leaf nodes reliably preserve BST ordering?

(i) Swap the values of the node to be deleted and the smallest leaf node with a larger value, then remove the leaf.

(ii) Swap the values of the node to be deleted with its predecessor or successor. If the predecessor or successor is a leaf, remove it. Otherwise, repeat the process.

(iii) If the node to be deleted does not have two children, simply connect the parent's child pointer to the node to the node's child pointer, otherwise, use a correct deletion strategy for nodes with two children.

(A) none
(B) *iii*
(C) all
(D) *i*
(E) *i* and *ii*
(F) *i* and *iii*
(G) *ii* and *iii*
(H) *ii*

Concept: heap shapes

229. In a heap, the upper bound on the number of leaves is:

(A) $O(1)$
(B) $O(\log n)$
(C) $O(n \log n)$
(D) $O(n)$

230. In a heap, the distance from the root to the furthest leaf is:

(A) $\theta(n)$
(B) $\theta(\log n)$
(C) $\theta(1)$
(D) $\theta(n \log n)$

231. In a heap, let d_f be the distance of the furthest leaf from the root and let d_c be the analogous distance of the closest leaf. What is $d_f - d_c$, at most?

(A) 1
(B) 0
(C) 2
(D) $\theta(\log n)$

232. What is the most number of nodes in a heap with a single child?

(A) 1
(B) 0
(C) $\Theta(\log n)$
(D) $\Theta(n)$
(E) 2

233. What is the fewest number of nodes in a heap with a single child?

(A) 2
(B) 0
(C) one per level
(D) 1

234. **T** or **F**: There can be two or more nodes in a heap with exactly one child.

235. **T** or **F**: A heap can have no nodes with exactly one child.

236. **T** or **F**: All heaps are perfect trees.

237. **T** or **F**: No heaps are perfect trees.

238. **T** or **F**: All heaps are complete trees.

239. **T** or **F**: No heaps are complete trees.

240. **T** or **F**: A binary tree with one node must be a heap.

241. **T** or **F**: A binary tree with two nodes and with the root having the smallest value must be a min-heap.

242. **T** or **F**: If a node in a heap is a right child and has two children, then its sibling must also have two children.

243. **T** or **F**: If a node in a heap is a right child and has one child, then its sibling must also have one child.

Concept: heap ordering

244. In a min-heap, what is the relationship between a parent and its left child?

(A) the parent has a smaller value
(B) the parent has the same value
(C) there is no relationship between their values
(D) the parent has a larger value

245. In a min-heap, what is the relationship between a left child and its sibling?

(A) the right child has a larger value
(B) the left child has a smaller value
(C) there is no relationship between their values
(D) both children cannot have the same value

246. **T** or **F**: A binary tree with three nodes and with the root having the smallest value and two children must be a min heap.

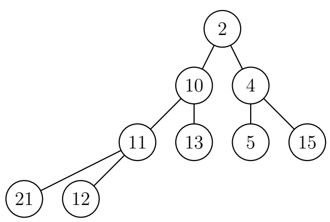
247. **T** or **F**: The largest value in a max-heap can be found at the root.

248. **T** or **F**: The largest value in a min-heap can be found at the root.

249. **T** or **F**: The largest value in a min-heap can be found at a leaf.

Concept: heaps stored in arrays

250. How would this heap be stored in an array?



(A) `[2,10,11,21,12,13,4,5,15]`
(B) `[21,11,12,10,13,2,5,4,15]`
(C) `[2,4,5,10,11,12,13,15,21]`
(D) `[2,10,4,11,13,5,15,21,12]`

251. Printing out the values in the array yield what kind of traversal of the heap?

(A) in-order
(B) level-order
(C) post-order
(D) pre-order

252. Suppose the heap has n values. The root of the heap can be found at which index?

(A) 0
(B) 1
(C) n-1
(D) n

253. Suppose the heap has n values. The left child of the root can be found at which index?

(A) 2
(B) 1
(C) n
(D) n-1
(E) n-2
(F) 0

254. Left children in a heap are stored at what kind of indices?

(A) all even but one
(B) all even
(C) a roughly equal mix of odd and even
(D) all odd but one
(E) all odd

255. Right children in a heap are stored at what kind of indices?

(A) all odd
(B) all odd but one
(C) all even but one
(D) a roughly equal mix of odd and even
(E) all even

256. The formula for finding the left child of a node stored at index i is:

(A) $i * 2 + 2$
(B) $i * 2$
(C) $i * 2 - 1$
(D) $i * 2 + 1$

257. The formula for finding the right child of a node stored at index i is:

(A) $i * 2 + 1$
(B) $i * 2$
(C) $i * 2 + 2$
(D) $i * 2 - 1$

258. The formula for finding the parent of a node stored at index i is:

(A) $(i + 1)/2$
(B) $(i + 2)/2$
(C) $(i - 1)/2$
(D) $i/2$

259. If the array uses one-based indexing, the formula for finding the left child of a node stored at index i is:

(A) $i * 2 + 1$
(B) $i * 2$
(C) $i * 2 + 2$
(D) $i * 2 - 1$

260. If the array uses one-based indexing, the formula for finding the left child of a node stored at index i is:

(A) $i * 2 + 1$
(B) $i * 2 - 1$
(C) $i * 2$
(D) $i * 2 + 2$

261. If the array uses one-based indexing, the formula for finding the parent of a node stored at index i is:

(A) $(i + 2)/2$
(B) $(i + 1)/2$
(C) $i/2$
(D) $(i - 1)/2$

262. Consider a trinary heap stored in an array. The formula for finding the left child of a node stored at index i is:

(A) $i * 3 - 2$
(B) $i * 3 + 1$
(C) $i * 3 + 2$
(D) $i * 3 - 1$
(E) $i * 3 + 3$
(F) $i * 3$

263. Consider a trinary heap stored in an array. The formula for finding the middle child of a node stored at index i is:

(A) $i * 3 - 2$
(B) $i * 3 + 3$
(C) $i * 3 + 1$
(D) $i * 3 - 1$
(E) $i * 3$
(F) $i * 3 + 2$

264. Consider a trinary heap stored in an array. The formula for finding the right child of a node stored at index i is:

(A) $i * 3 + 3$
(B) $i * 3 + 2$
(C) $i * 3 - 1$
(D) $i * 3 - 2$
(E) $i * 3$
(F) $i * 3 + 1$

265. Consider a trinary heap stored in an array. The formula for finding the parent of a node stored at index i is:

(A) $(i - 2)/3$
(B) $(i - 1)/3$
(C) $(i + 1)/3$
(D) $(i + 2)/3$
(E) $i/3 + 1$
(F) $i/3 - 1$

Concept: *heap operations*

266. In a max-heap, the minimum value can be found in time:

(A) $\theta(n)$
(B) $\theta(n \log n)$
(C) $\theta(\log n)$
(D) $\theta(1)$

267. Suppose a min-heap with n values is stored in an array a. In the *extractMin* operation, which element immediatelyreplaces the root element (prior to this new root being sifted down).

(A) the minimum of `a[1]` and `a[2]`
(B) `a[1]`
(C) `a[2]`
(D) `a[n-1]`

268. The *findMin* operation takes how much time?

(A) $\Theta(1)$
(B) $\Theta(n \log n)$
(C) $\Theta(\log n)$
(D) $\Theta(n)$

269. The *extractMin* operation takes how much time?

(A) $\Theta(n)$
(B) $\Theta(n \log n)$
(C) $\Theta(1)$
(D) $\Theta(\log n)$

270. Merging two heaps of size n and m, $m < n$ takes how much time?

(A) $\Theta(\log n + \log m)$
(B) $\Theta(n \log m)$
(C) $\Theta(n * m)$
(D) $\Theta(\log n * \log m)$
(E) $\Theta(m \log n)$
(F) $\Theta(n + m)$

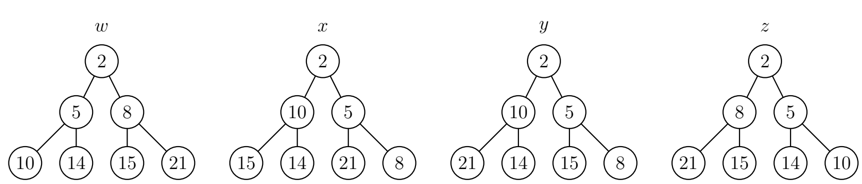
271. The *insert* operation takes how much time?

(A) $\Theta(1)$
(B) $\Theta(n \log n)$
(C) $\Theta(n)$
(D) $\Theta(\log n)$

272. Turning an unordered array into a heap takes how much time?

(A) $\Theta(n \log n)$
(B) $\Theta(1)$
(C) $\Theta(\log n)$
(D) $\Theta(n)$

273. Suppose the values 21, 15, 14, 10, 8, 5, and 2 are inserted, one after the other, into an empty *min*-heap. What does the resulting heap look like? Heap properties are maintained after every insertion.



(A) z
(B) w
(C) x
(D) y

274. Using the standard *buildHeap* operation to turn an unordered array into a *max*-heap, how many parent-child swaps are made if the initial unordered array is `[5,21,8,15,25,3,9]`?

(A) 5
(B) 6
(C) 7
(D) 2
(E) 4
(F) 3